DOCUMENT RESUME

ED 260 688 IR 011 760

AUTHOR Harvey, Wayne
TITLE Designing Educational Software for Tomorrow.
INSTITUTION SRI International, Menlo Park, Calif.
PUB DATE May 85
NOTE 12p.
PUB TYPE Viewpoints (120)

EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS Artificial Intelligence; Cognitive Style; *Courseware; Design Requirements; *Environmental Influences; *Instructional Design; Instructional Development; *Learning Strategies; *Microcomputers; Quality Control; Teaching Methods
IDENTIFIERS *Software Design

ABSTRACT

Designed to address the management and use of computer software in education and training, this paper explores both good and poor software design, calling for improvements in the quality of educational software by attending to design considerations that are based on general principles of learning rather than specific educational objectives. This approach also focuses attention on the student-computer segment of the wider context in which learning occurs and which includes the external environment, e.g., the school, the home, and cultural institutions, as well as other individuals, e.g., parent, teacher, fellow students, who affect the interactions between the student and the computer. Topics discussed include: (1) goals for computer use in education; (2) the software design process and the source of quality productions; (3) student orientation to software; (4) student interaction with hardware and software; (5) ways to facilitate student understanding; (6) promising directions to pursue; and (7) the Advanced Instructional Technology Program at SRI International. Included are diagrams which illustrate the concepts of software types and learning skills, educational software development, design considerations for educational software, and the interaction of computer science and cognitive engineering. The paper concludes with a call for instructional designers to use information on how people store, manipulate, and process knowledge in the design of educational software. (JB)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED260688

U.S. DEPARTMENT OF EDUCATION
NATIONAL INSTITUTE OF EDUCATION
EDUCATIONAL RESOURCES INFORMATION
CENTER (ERIC)

X This document has been reproduced as received from the person or organization originating it.

☐ Minor changes have been made to improve reproduction quality.

• Points of view or opinions stated in this document do not necessarily represent official NIE position or policy.

DESIGNING EDUCATIONAL SOFTWARE FOR TOMORROW

Wayne Harvey

SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025
(415) 859-4004

BEST COPY AVAILABLE

May 1985

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY
Wayne Harvey

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

IR011760

The potential for computers to improve educational is enormous—more dramatic than any invention since writing. Yet that potential is not being met. Simply put, our schools are being swept up in a tidal wave of technology without any idea of how to make wise use of it.

Representative Albert Gore, Jr.
Author of the National Education
Software Act of 1984

Despite the vast quantity of educational software available for microcomputers, there are few outstanding programs that use the computer as an effective and innovative educational tool. Too often software designers focus on the technical qualities of their program implementation rather than attending to the kind of learning experience that should be created.

To realize the potential of these new technologies, the educational software of the future will have to work with the hardware to become an "intelligent" partner in learning rather than an entertaining but inflexible taskmaster. This paper considers where we are and what is needed, and suggests some promising directions to pursue.

Many of the expectations that educators have for the uses of computers to mediate instruction are based on the educational software predominant in schools and homes today. Unfortunately, the computer today is an underused and often misused educational tool, with even the best educational software barely tapping its potential. If we are going to see great improvements in the quality of educational software, we must all be more aware of the design considerations that make for quality in software and demand that developers' products be sensitive to these considerations. Much of what we know about good software design and good instruction often is not reflected in present-day computer-assisted instruction. Perhaps this would not be so if we expected more of software designers.

Before exploring good and poor software design in this paper, it is necessary to develop a perspective on the possibilities for using computers as educational tools. What are reasonable goals for computer use in education? After answering that question, the software design process is examined and criteria for quality in instructional software are identified. Finally, we will look at reasonable expectations for our future instructional systems. How will current research efforts contribute to advanced computer-assisted instruction, and what are promising directions to be pursuing?

The approach in this paper is to focus attention on the student-computer segment of a wider context in which learning occurs. The wider context includes the external environment (e.g., the school, the home, or a museum) in which the learning takes place, as well as other individuals (e.g., the teacher, a parent, or other students) who affect the interactions between student and computer. Although successful educational software design requires consideration of these environmental and sociological factors, we can still learn about designing quality into software by examining the more easily handled, but still very complex, student-computer relationship.

Goals for Computer Use in Education

With many thousands of pieces of educational software available for microcomputers, it is no wonder that people can make little sense out of how to use computers to help students learn. Much of this software is "not-very-fancy" drill and practice activities, and one often hears that "there's almost no good educational software; it's mostly drill and practice." But behind this statement lies an unfortunate confusion between quality of the software and complexity of the educational objective.

Obviously, there are many different things to learn, and there are many different ways to learn things. What is important is to match what is to be learned—which may in fact be something very mundane—with the best way to learn it. So if our objective is to teach some high-level problem-solving skills—yes, drill and practice software may very well be "bad" software. But, if we want a child to learn the alphabet, this type of software may be ideal. Then the question becomes, "is it high-quality drill and practice software or is it poorly designed?"

Proponents of computer use in education often make the mistake of promoting for general use some particular kind of software that might be useful mainly for achieving specific learning objectives. One cannot specify the "right" way to use a computer any more than the "right" way to use a book. A computer is a general-purpose tool and as such can be used to accomplish different objectives. If one wants to strengthen creative skills, maybe certain games or puzzles will be the appropriate kind of software while a focus on logical thinking skills might be better handled by creative simulations or learning to write programs (see Figure 1, "Software Types and Learning Skills").

A better way to think about the use of computers in education is to consider general principles of

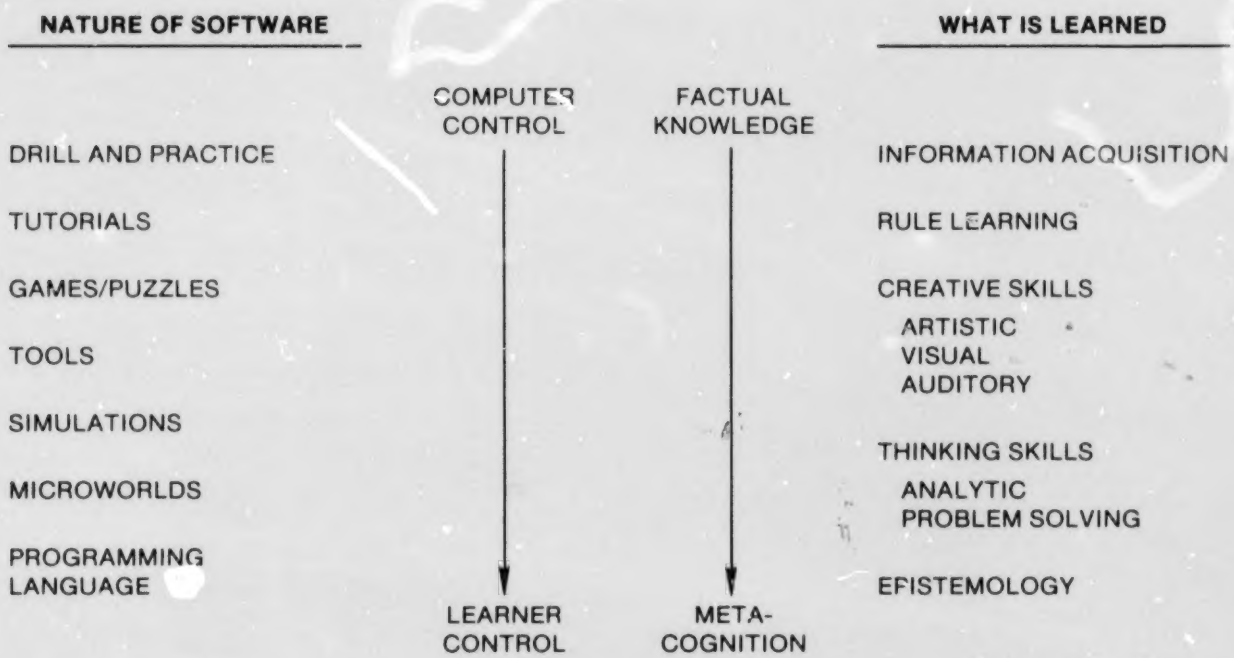


Figure 1 SOFTWARE TYPES AND LEARNING SKILLS

learning rather than specific educational objectives. Instead of focusing on what kinds of *software* should be created, one can gain new perspectives and insights by considering what kinds of *learning experiences* should be created. Much of the power of a computer as an educational tool lies in its ability to let the student *interact* individually with ideas and information in ways that facilitate learning. Sadly, a large portion of existing computer-assisted instruction fails to meet this very basic goal!

For each type of software, we should expect the rule, not the exception, to be that the product is stimulating, challenging, and enjoyable—in short, intrinsically motivating. Educational software should be designed to encourage creative and independent exploration and acquisition of knowledge or skills. In summary, the goals of using computers as educational tools should be defined in terms of learners. Is there interaction? Do the learners have control over the material and the presentation? Are they motivated? It is in these goals that we begin to identify the source of real quality in educational software.

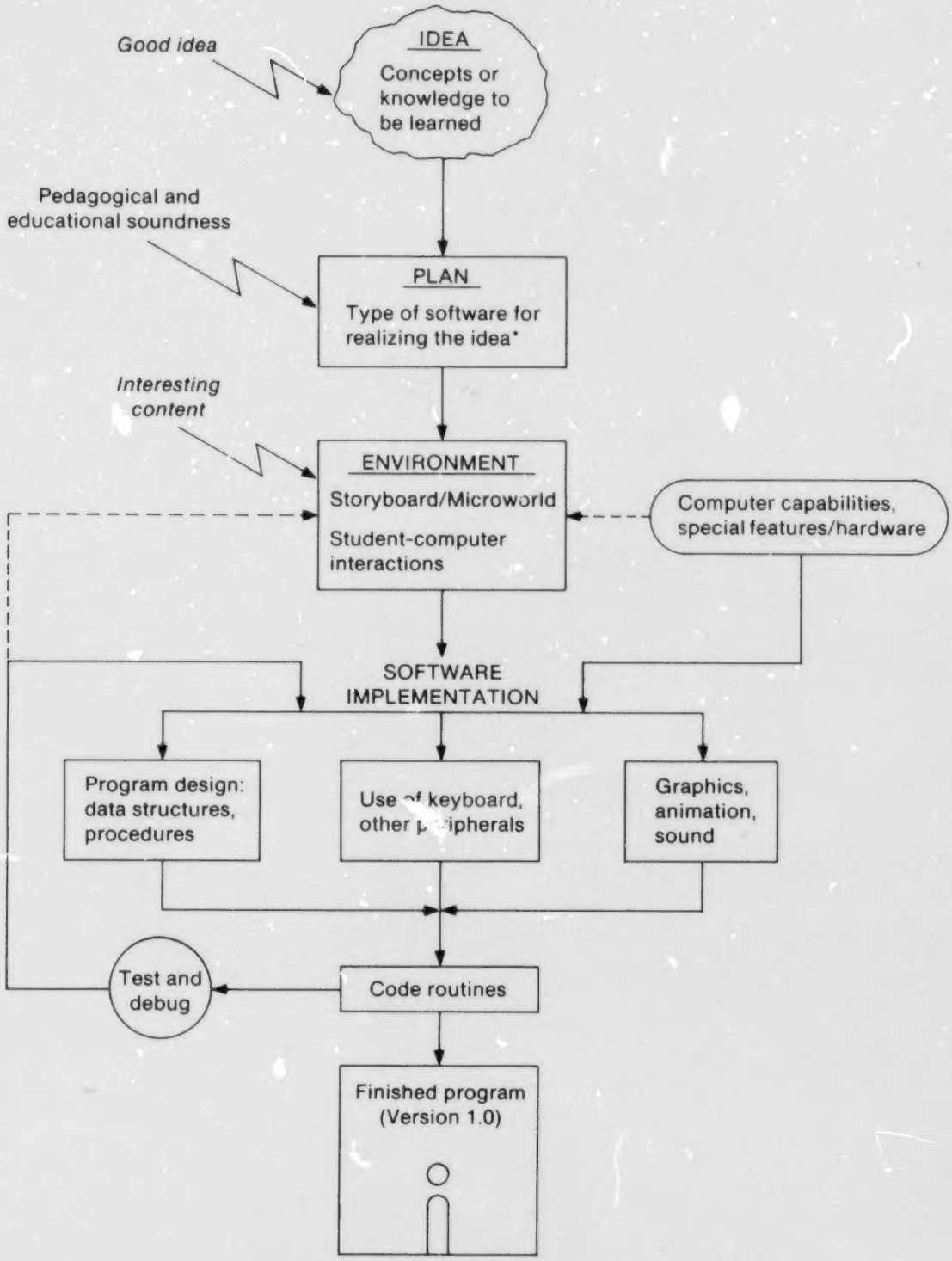
The Software Design Process—Where Is the Source of Quality?

It is not surprising that among the vast quantity of educational software available for microcomputers, there are few outstanding programs that utilize the computer as an effective and innovative educational tool. Creating high-quality educational software requires an unusual combination of expertise in diverse areas, especially pedagogical design, software design, subject content, and artistic abilities. The programming wizard who sets out to create instructional software often fails to understand the importance of good instructional design. And the educator who tries to implement a good idea often has little understanding of how to use the computer effectively as a tool.

It is only a partial solution to have instructional designers and programmers working together in an attempt to apply each area of expertise to the problem of educational software design and development. In fact, the development process cannot be effectively divided into separate areas of understanding without reducing the quality of the results. Rather, our best educational software is going to be produced by software designers with multidisciplinary training that includes instructional design, computer science, and cognitive psychology. The need for such combined expertise will be evident from a review of the educational software design process.

One expects the design of good educational software to begin with an idea for certain concepts or a kind of knowledge to be learned (see Figure 2, "Educational Software Development"). It is amazing that some educational software is designed without starting with any educational objectives, let alone a good idea. Sometimes the result will be software with powerful learning possibilities, but more often the software fails to promote any real educational objectives.

Early on, a decision must be made as to the type of software that would be most effective for the learning to be achieved (see Figure 1). This leads directly to consideration of the software environment—the kind of system with which the student will interact and an explicit description or specification of those interactions. It is at this point where a wide range of skills and experience must come together to produce the best results, and so it is here where the lack of combined expertise is so costly to the quality of design. In defining the environment, one must understand what promotes and what hinders the learning process—in particular, the types of interactions that should be created between the computer and the student. The educational content of the computer-assisted instruction and the technical capabilities of the system on which the software is to run also have to be considered.



*See Figure 1, "Software Types and Learning Skills."

Figure 2 EDUCATIONAL SOFTWARE DEVELOPMENT

All of this leads to the actual software implementation, which requires descriptions of: data structures and algorithms for creating the computer world or desired effects, specification of how best to use the keyboard or other input devices, and conceptualization of meaningful graphics, animation, or sound. Even in this highly technical activity, the need for combined expertise is clear. Finally one is ready to write, test, and modify the program. Many revisions are required before good educational software can become great educational software.

Designing good software is a very difficult process, and it will probably remain so for the foreseeable future. But often it would have been no more difficult to improve the quality of a piece of poorly designed software if only the developer had been more aware of sometimes subtle, but critically important issues. These include providing orientation for the student, creating interactions between the student and the computer that stimulate the learning process, and presenting materials in ways that enrich the student's understanding (see the box, "Design Considerations for Educational Software").

Student Orientation

Orienting the student in the most simple sense is extremely easy to do and yet is often neglected. Providing clear instructions is a beginning, but we should expect the software to provide some help if the user requests it, especially when the user does something wrong. Furthermore, if a student continues to have difficulty with a task, he or she should be given an opportunity to work on a simpler task. That is, there should be levels of difficulty, selected either by or for the user. Finally, users should know "where they are" in the software. There should be frames of reference that help guide students through their interaction with the software. This becomes especially important when there are different possible modes one can be in or when there are nested menus of options.

DESIGN CONSIDERATIONS FOR EDUCATIONAL SOFTWARE

STUDENT ORIENTATION

- Clear instructions. On-line help.
- Good error handling.
- Levels of difficulty matched with skill levels.
- Recognizable frame of reference.

STUDENT INTERACTION

- Clear presentation of choices, actions.
- Easy and memorable selection of choices, actions.
- Feedback provided when user acts/doesn't act.

STUDENT UNDERSTANDING

- Consistent presentation format, use of screen.
- Appropriate graphics, sound, text:
  - — Enhance ideas
  - — Useful metaphors
  - — Non-distracting.

Student Interaction

Much work has been directed at exploring important principles of human-machine interaction, and this takes on critical importance when designing educational software. Yet it is not unusual to find that more time is spent learning *how* to interact with a program than learning *from* interacting with the program. When the user has to make a choice, the options should be clearly presented and the actions that will result from choosing each option should be clear. Furthermore, the ways to select and activate an

option should be obvious and easy (e.g., move a cursor over the item or point to the item, press a certain key or press a mouse button). Finally, whenever the user does something, there should be an immediate indication that the computer recognized the action—something should happen. Not only that, but if the user is supposed to do something and doesn't, after some period of time the software should provide some suggestion, warning, reminder, or other help.

Student Understanding

Such feedback must be done with care because it must not get in the way of what is to be learned and distract the learner: the student's understanding is the ultimate goal. How information is presented on the screen—where, for how long, in what colors, and in what arrangement—will significantly affect the student's learning of the material. In particular, consistency leads to understandability. If helpful hints are presented on the bottom of the screen, they should always be on the bottom of the screen. If graphic icons are used for a variety of purposes, their location on the screen should help the user identify their type of use. Screen location, color, and size can all be used to convey significant meaning; too often they appear to be selected randomly and used inconsistently.

Many software designers fail to consider how to use graphics, sound, and text effectively to strengthen the learning process. Fancy graphics, colors, and other effects in themselves do not mean improved learning. Graphics designed to reinforce the concepts being taught are much more valuable than spectacular effects that merely reward correct answers. Graphics should be used to enhance the idea, to enliven it, to make it more understandable. It is sad when software designers can do no better than create graphics that are used as "a spoonful of sugar to help the medicine go down."

Promising Directions to Pursue

The design criteria discussed above are all suggestions that any developer could incorporate into a software design process immediately to improve the educational quality of software. If more emphasis were given to the concerns raised in this paper, there would be dramatic improvements in the quality of available software. But this is only a first step; we should expect much more in the future from computer systems designed to help people learn.

The unique power of the computer is its potential for truly individualizing instruction, not simply by providing different difficulty levels, but by "understanding" the user of the system. Different people have different learning styles and learn more easily under different circumstances. We should expect in the future software that adapts to these differences. The computer should also respond "intelligently" to a student's input, so that it is more a partner in learning and less a mechanical device.

Some individuals will benefit more by highly structured environments in which learning is very directed, while others can learn much more through discovery. Software needs to "know" when to present material, when to interact, when to take control, and when to give the learner control. It needs to show tolerance for individual creativity and individual differences in problem-solving approaches and other high-level thinking tasks. And it should, as a good teacher does whenever possible, recognize what the user *intended* to mean, say, or do, rather than making literal and inflexible interpretations.

Clearly we do not yet have a deep enough understanding of how people think, learn, and communicate to fill these requests. Some of the current research efforts in artificial intelligence and cognitive psychology, however, will bring us closer to these accomplishments. We need to learn how to model

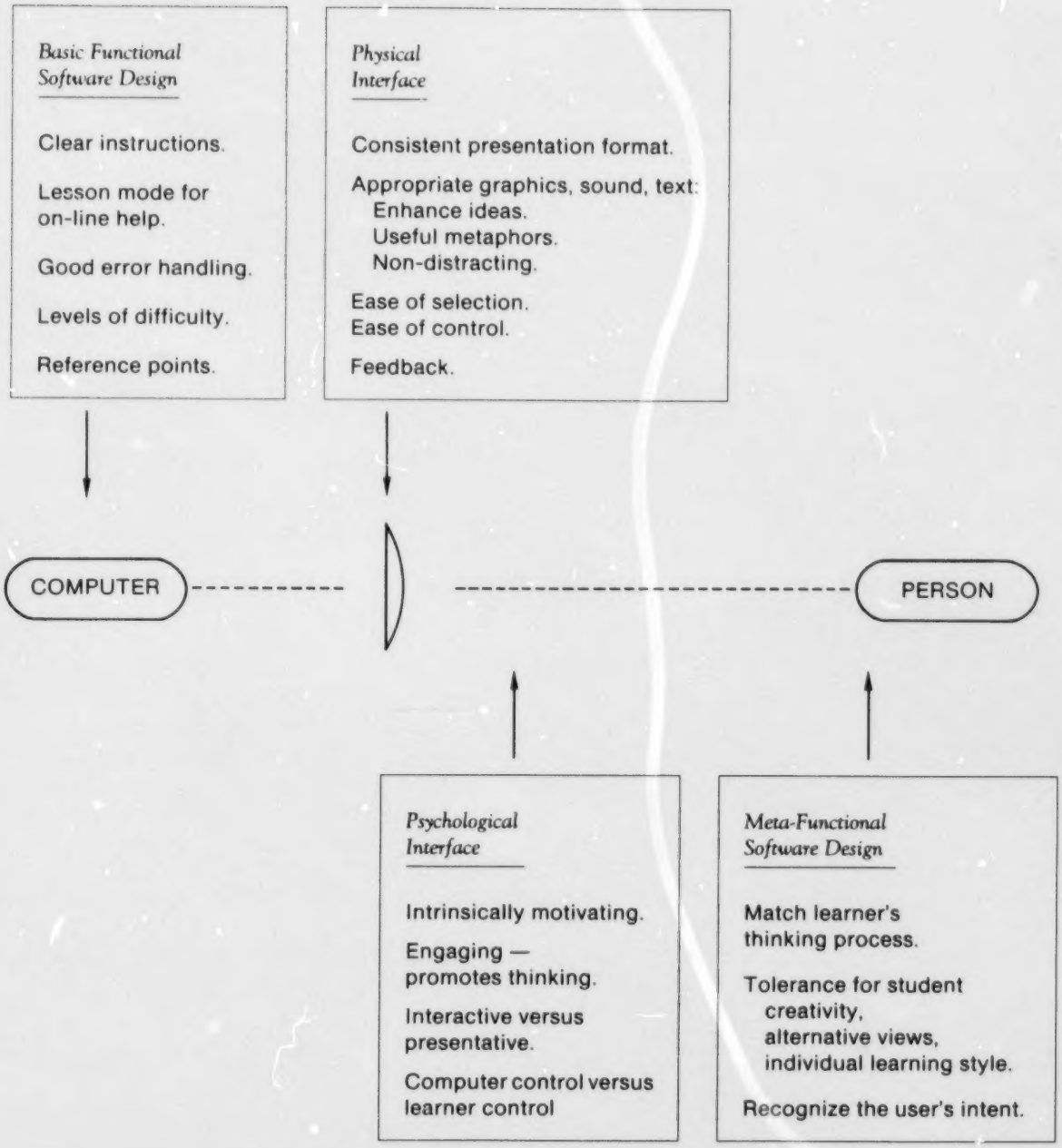


Figure 3 FROM COMPUTER SCIENCE TO COGNITIVE ENGINEERING

the student's understanding, attitudes, style, and state of mind.

In the past (and too often in the present) we have looked to computer scientists to focus their attention on the computer and design quality into software. Now, if we are to make full use of the computer as an educational tool, we must focus on the person, learn about the learner, and look to this side of the computer-student system to advance the state of the art (see Figure 3, "From Computer Science to Cognitive Engineering").

Currently our expectations for acceptable educational software usually require only that it meet basic functional design criteria, and even those criteria are rarely met. Sometimes we will intuitively feel that a piece of software is exceptional. Usually, this impression is the result of quality designed into the physical interface. This, however, is not enough! For significant learning to result from student-computer interactions, we must begin to explore and attend to the psychological interface. And in the near future we should expect even more. Today we can use our understanding of how microcomputers store, manipulate, and process information to help us design high-quality software. Tomorrow we must use knowledge of how *people* store, manipulate, and process knowledge to design powerful learning tools.

ADVANCED INSTRUCTIONAL TECHNOLOGY PROGRAM
SRI International

The Advanced Instructional Technology Program is a new multidisciplinary capability being developed at SRI International to address the management and use of new technologies in education and training. The program's missions are 1) to pursue research efforts needed to understand how to effectively use new tools—including microcomputers, videodiscs, and telecommunications—to enhance learning, 2) to develop software and hardware technologies to demonstrate the educational possibilities, and 3) to evaluate and help integrate these learning tools into educational environments.

The new and more powerful electronic technologies being developed have the potential to revolutionize instruction. Microcomputers are coming into wide use as educational and training tools, and telecommunications and videodisc technologies are also finding their place in the instructional arena. Unfortunately, the growing availability of the hardware itself will not necessarily improve the education process.

Technology has the potential to expand the repertoire of traditional instructional presentations, but to achieve efficiency and effectiveness we must develop a better understanding of how technologies mediate learning. The promise for the future is the development of integrated systems capable of interacting intelligently and sensitively with learners and capable of providing a responsive, structured environment in which various skills and concepts can be learned, developed, and practiced. The Advanced Instructional Technology Program is bringing together the multidisciplinary talents of educators, instructional designers, cognitive psychologists, computer scientists, and engineers to further explore these possibilities.

SRI International is an independent, nonprofit corporation performing a broad spectrum of problem-oriented research under contract to government, industry, and business. SRI serves clients in all parts of the United States and throughout the world.

For more information about the Advanced Instructional Technology Program, contact:

Wayne Harvey
SRI International
333 Ravenswood Avenue, Room BS319
Menlo Park, CA 94025
(415) 859-4004